VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019

Martin Krybus

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Krybus**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ATLAS consulting spol. s r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **RNDr. Eliška Ochodková, Ph.D.**

Konzultant bakalářské práce: Mgr. Tomáš Řehák

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 19. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 19. dubna 2019

ATLAS consulting spol. s r.o.
Výstavní 292/13
702 00 Ostrava, Moravská Ostrava
IČO 46578706, DIČ CZ46578706

.....

Rád bych poděkoval vedoucímu vývoje Mgr. Tomáši Řehákovi ve společnosti ATLAS consulting spol. s r.o. za profesionální vedení a cenné rady během absolvování praxe. Dále bych rád poděkoval RNDr. Elišce Ochodkové, Ph.D. za vedení práce a všem kolegům ze společnosti, s kterými jsem měl možnost spolupracovat.

Abstrakt

Cílem této závěrečné práce je dokumentace průběhu individuální odborné praxe, která proběhla ve společnosti ATLAS Consulting spol. s.r.o. Náplň mé práce spočívala ve vývoji backendu nejkomplexnějšího právního informačního systému v jazyce Java. V tomto dokumentu se hlavně zaměřuji na popis a zvolený postup řešení jednotlivých úkolů. Dále popisuji znalosti získané během studia uplatněné v praxi a absenci znalostí, které mi v praxi scházeli. V závěru se věnuji celkovému hodnocení praxe a dosaženým výsledkům.

Klíčová slova: Java, informační systém, REST, Cucumber, webová aplikace, Spring Boot, Git, Mockito Framework, IntelliJ IDEA, GraphQL

Abstract

The aim of this thesis is the documentation of the course of individual professional practice, which took place in the company ATLAS Consulting spol. s.r.o. My job was to develop the backend of the most comprehensive legal information system in Java. In this document, I mainly focus on the description and the chosen procedure for solving individual tasks. Then I describe the knowledge gained during the study applied in practice and the absence of knowledge that I missed in practice. In the conclusion I deal with the overall evaluation of practice and achieved results.

Key Words: Java, Information System, REST, Cucumber, Web Application, Spring Boot, Git, Mockito Framework, IntelliJ IDEA, GraphQL.

Obsah

Seznam použitých zkratek a symbolů

PDF	–	Portable Document Format
SQL	–	Structured Query Language
XML	–	Extensible Markup Language
HTML	–	Hyper Text Markup Language
REST	–	Representational State Transfer

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Nedílnou součástí výuky na vysoké škole je absolvování individuální praxe. Zde má student možnost uplatnit teoretické a praktické znalosti získané během studia. V průběhu této praxe student získává zkušenosti se skutečným vývojem ve firmě a učí se týmové práci.

Rozhodl jsem se absolvovat individuální odbornou praxi ve společnosti ATLAS Consulting s.r.o. jako Java backend developer. Firma se věnuje vývoji, výrobě a distribuci informačních systémů. Nyní se intenzivně zabývá komplexním právním informačním systémem Codexis Green.

V první části jsou popsány základní informace o firmě, její zaměření a mou pracovní pozici v rámci odborné praxe. V druhé části bakalářské práce popisuji zadané úkoly a řešení jednotlivých úkolů. V závěru práce se věnuji znalostem získaným během studia, které jsem uplatnil při absolvování odborné praxe a znalostem, které mi v průběhu praxe naopak scházeli.

2 Firma ATLAS Consulting spol. s.r.o. a pracovní zařazení studenta

Firma ATLAS Consulting s.r.o. [2] je ryze českou společností, která se nachází v Ostravě na ulici Výstavní 292/13 a už od roku 1992 se věnuje vývoji, výrobě a distribuci informačních systémů. Do těchto systémů patří manažerské systémy Econix a právní informační systémy Codexis.

Do systémů Codexis patří Codexis Green, Codexis Mobile, Codexis Advokacie, Codexis Academia, Codexis Justice, Commentis, Advokátní spis, Právní výpočty a Právní kalkulačka. Codexis Green je webová aplikace navržená pro snadnou práci v právním sektoru. Codexis Mobile je mobilní aplikace s omezenými funkcemi. Slouží k nahlédnutí do právních předpisů, vyhlášek a zákonů. Codexis Advokacie se zaměřuje na související dokumenty, integraci práva a související judikatury. Speciálně pro studenty byl navržen Codexis Academia. Soudní oblastí se zabývá systém Codexis Justice. Systém Commentis umožňuje uživatelům přistoupit k unikátnímu souboru plnotextových online komentářů s judikaturou. Jednoduchou a rychlou administrativu v právním sektoru zajišťuje software Advokátní spis.

Do systémů Econix patří Equanta, Smlouvy, Manažer datových stránek, Daňová kancelář, Kontrola Insolvence, Navigal, Evidencia zmlúv. Software Equanta byl navrhnut speciálně pro finanční analýzu a plánování. Systém Smlouvy nabízí efektivní práci se smlouvami a umožnuje uživateli přehlednou evidenci smluv, ke které může například přidávat poznámky. Program Kontrola insolvence zajišťuje vždy přehled o situaci vašich partnerů, dodavatelů a zákazníků. Pro uživatele vlastnící více datových schránek byl vyvinut Manažer datových schránek, který po přihlášení do aplikace se automaticky přihlásí do všech dalších datových schránek.

Pracovně jsem byl zařazen na pozici Java backend developer pro Codexis Green, což je právní informační systém v online verzi. Mým úkolem bylo zdokonalování a rozšiřování již implementovaných částí projektu a vývoj nové funkcionality, která většinou znamenala vytvoření nové služby nebo přidání aplikačního endpointu pro zisk dat z backendu. Nedílnou součástí mé pracovní náplně také bylo udržování kódů včetně oprav chyb v systému na základě dat získaných testovacím oddělením.

3 Úkoly zadané v průběhu praxe

Během mé praxe jsem dostával úkoly pro právní systém Codexis Green, mezi které patřil složitý a časově nejnáročnější úkol. Výsledkem tohoto úkolu bylo umožnit uživatelům zobrazení novelizací paragrafů přes všechna časová znění. Následujícím úkolem bylo rozšířit funkcionalitu sledovaných dokumentů o možnost sledování jen specifických částí dokumentu. Dalším úkolem bylo zajistit funkci, která měla vyexportovat dokument do formátu DOCX nebo PDF a umožnit tak uživateli jej stáhnout. Kromě těchto náročnějších úkolů jsem dostával i lehčí úkoly zahrnující například roztřídění výstupních dat do složek, vytvoření interaktivní knihovny komentářů, opravy chyb či refaktoring stávajícího kódu.

3.1 Zobrazení novelizací k paragrafům

Úkolem bylo rozšířit systém o značky u jednotlivých částí dokumentu, které indikují, zda se daný paragraf změnil (kdykoliv v minulosti) nebo že se změní (kdykoliv v budoucnosti). Momentálně v aplikačním serveru již byla implementována funkcionalita, která dokáže srovnat předchozí a budoucí znění a je tudíž schopná detekovat změnu paragrafu o jedno znění zpět nebo vpřed. Nicméně je to funkcionalita časově náročná a za běhu aplikace by bylo téměř nemožné porovnat všechny znění dokumentu a tím detekovat změněné části dokumentu přes veškerá znění. Nad zadaným úkolem jsem strávil přibližně 30 dní.

3.2 Export dokumentu do formátu DOCX a PDF

Úkolem bylo najít způsob, jak dokument vyexportovat do formátu DOCX při zachování formátování a následně převést formát DOCX do formátu PDF. Navíc export dokumentu musel umožňovat použití SectionPredicate, což je funkce, která umožnuje omezení rozsahu exportu na vybrané části dokumentu. Časová náročnost tohoto úkolu byla 20 dní.

3.3 Sledované dokumenty - možnost volby sledovaných částí dokumentu, volby notifikací

Tento úkol byl časově nejméně náročný. Vypracování úkolu mi trvalo přibližně 15 dní. Stávající logika systému umožňovala uživatelům sledovat změny jen k celému dokumentu, které si uživatel mohl nechat později zaslat na email. Mezi tyto změny například patří změna souvisejících dokumentů k části dokumentu nebo změna textu v paragrafu. Účelem bylo upravit sledované dokumenty tak, aby si uživatel:

- mohl zvolit, v jaké části dokumentu chce sledovat změny
- mohl zvolit, k jakým částem dokumentu a jaké typy souvisejících dokumentů chce sledovat (literatura, rekodifikace, judikatura apod.)
- mohl zvolit, k čemu a kdy chce dostávat emailové notifikace (změny textů, související změny)
- mohl zvolit, s jakým předstihem chce být o změnách částí dokumentu informován

4 Řešení zadaných úkolů

Všechny úkoly jsem implementoval ve vývojovém prostředí IntelliJ IDEA [5] od JetBrains s.r.o. v programovacím jazyce Java [3]. Pro přípravu dat byla využívána technologie Apache Groovy [4]. Přenos infomací mezi frontendem a backendem byl obstaráván pomocí dotazovacího jazyka GraphQL [6]. Pro zpřístupnění uživateli stahovat data z prohlížeče jsem použil REST [7] endpointy. Jednotlivé implementace úkolů jsem rozvrhl na menší funkce, tak aby se dali jednoduše otestovat pomocí unitových testů. Pro testování koncových bodů služeb byly použity Cucumber [8] testy.

4.1 Zobrazení novelizací k paragrafům

Cílem tohoto úkolu bylo rozšířit dokumenty právního informačního systému o funkci zobrazující změny paragrafů mezi jednotlivými časovými úseky. U jednotlivých částí dokumentu měly vzniknout šipky, které při kliknutí zobrazí uživateli všechna znění měnící daný paragraf. (viz obrázek 1). Šipka směrem do leva indikovala změnu v minulosti. Šipka směrem do prava indikovala změnu v budoucnosti. Vybráním znění se mělo zobrazit srovnání aktuálního znění části a vybraného znění části.

Obrázek 1: Příklad novelizací paragrafu 2

Aby bylo možné uživatelům zobrazit v dokumentu k jednotlivým paragrafům změny na časové ose, bylo potřeba nejdříve srovnání všech znění jednotlivých dokumentů. Samotný výpočet bez předpřipravených dat by byl za běhu velice časově náročný, vzhledem k tomu, že se v databázi nachází i dokumenty obsahující až 20 znění a samotná znění obsahují až 50 stran. Proto jsem tyto

informace předpočítal pomocí operace dodatečně před zapnutím aplikačního serveru. Operace se spouští vždy před nasazením nové verze aplikace nebo když oddělení právních systémů přidá do databáze nová data. V samotné implementaci, která měla za úkol předpočítat data, bylo potřeba pro každý dokument legislativy České republiky porovnat všechna časová znění a tyto indikované změny uložit do objektů identifikovatelných číslem dokumentu. Implementaci operace procházející všechny uložené dokumenty v datech a algoritmus identifikující změněné paragrafy pro dokument jsem oddělil. Začal jsem implementací malé funkce, která pro dvě konkrétní znění dokumentu vrátí změněné části dokumentu (viz výpis 1).

```
private List<DocPlace> getChangeContentItems(DocSection chosen, DocSection
    prevOrNext) {
    DocDiffResult docDiffResult = new TextLineDiffService(chosen, prevOrNext,
        ImmutableList.of(DiffViewType.FULL_DOC, DiffViewType.INLINE),
        imageResolveFc).get();
    return docDiffResult.getChangeContentItemsList().stream()
            .map(p -> new DocPlace().setPartPath(ImmutableList.of(new DocPart().
                setValue(p).setDocPart(DocPartType.SECTION_LINK))))
            .collect(Collectors.toList());
}
```

Výpis 1: Funkce getChangeContentItems

Znění mohou být derogační, novelizační, nulové, tečkové nebo fiktivní. Derogační a tečková znění se nedají porovnávat, tudíž takové znění budu v cyklu rovnou přeskakovat. Zdrojový kód funkce validující typ znění lze vidět ve výpisu 2.

```
private boolean notValidForDiff(LCRZneni2 zneni, LCRZneni2 nextZneni) {
        return zneni.getZneniType() == ZneniType.TECKOVE_ZNENI ||
               nextZneni.getZneniType() == ZneniType.TECKOVE_ZNENI ||
               zneni.getZneniType() == ZneniType.DEROGACE ||
               nextZneni.getZneniType() == ZneniType.DEROGACE;
    }
```

Výpis 2: Funkce notValidForDiff

Pokračuji další funkcí vracející pro libovolný dokument informaci o tom, v jakých zněních se dané paragrafy v dokumentu změnily (viz výpis 3).

```
public Map<DocPlace, List<LCRZneni2>> apply(Doc doc, CRMetadata metadata) {
    Map<DocPlace, List<LCRZneni2>> changes = new HashMap<>();
    List<Zneni> zneniList = metadata.getZneni();
    if (zneniList.size() < 2) { //zneniList needs at least 2 zneni
        return changes;
    }

    DocPlaceLocator latestDPL = new DocPlaceLocator(doc.viewLatest());
    for (int i = 0; i < zneniList.size() - 1; i++) {
        LCRZneni2 zneni = zneni2LcrZneniFc.apply(zneniList.get(i));
        LCRZneni2 nextZneni = zneni2LcrZneniFc.apply(zneniList.get(i + 1));
        if (notValidForDiff(zneni, nextZneni)) {continue;}
        LocalDate chosenViewDate = Dates.toLocalDate(zneni.getFrom());
        DocSection section = doc.view(chosenViewDate);
        DocSection nextDocSection = doc.view(doc.getNextDate(chosenViewDate));
        DocPlaceLocator dpl = new DocPlaceLocator(nextDocSection);

        if (i == 0) { //Adds nulove znění to all from znění 0
            List<DocPlace> nuloveZneniLinks = docSection2Links(section);
            addChangesToMap(nuloveZneniLinks, latestDPL, dpl, changes, zneni);
        }
        List<DocPlace> changedItems = getChangedItems(section, nextDocSection);
        addChangesToMap(changedItems, latestDPL, dpl, changes, nextZneni);
    }
    changes.entrySet().removeIf(entry -> entry.getValue().size() < 2);
    return changes;
}
```

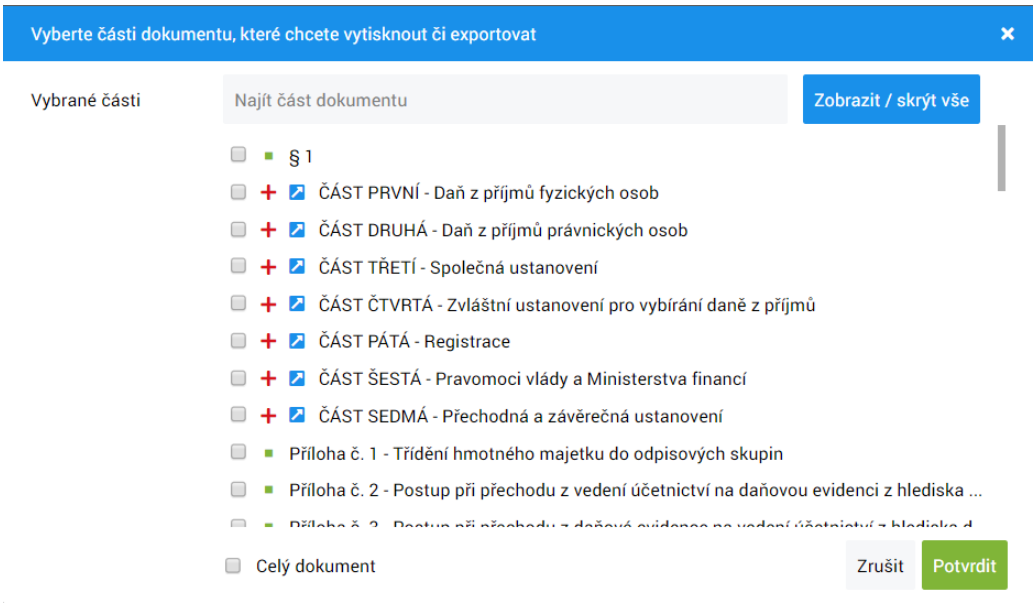
Výpis 3: Funkce identifikující změněné paragrafy

Následuje implementace a rozšíření služeb. Jako první jsem vytvořil Cucumber test s očekávaným výstupem pro ladění. Do již vytvořené služby, která se starala o tvorbu XML dokumentu, jsem přidal logiku pro identifikaci paragrafů mající alespoň jednu změnu v minulosti nebo v budoucnosti zobrazením šipky u jednotlivých částí dokumentu. Pokud se daný paragraf změnil v budoucnosti i v minulosti, měla být zobrazena šipka do leva i šipka do prava zároveň. Vytvořil jsem službu DocPlaceChangesTimelineService starající se o vrácení časových řezů upravující dané znění paragrafu. Vstupním parametrem této služby je identifikační číslo aktuálně otevřeného dokumentu a časové znění, které má uživatel otevřené. Aby služba byla pro frontend

dostupná a mohl ji zavolat, tak byl přidán GraphQL endpoint, který definuje jak se bude k službě přistupovat. Při volání samotné služby jsem načetl předpočítaná data podle vstupních parametrů a vrátil časová znění měnící daný paragraf na výstup uživateli.

4.2 Export dokument do formátu Word a PDF

Jako uživatel chci mít možnost převést stávající dokument do lehce přenositelného formátu. Úkolem bylo implementovat funkci, která převede dokument do Microsoft Word a PDF dokumentu. Export dokumentu měl také umožňovat použití SectionPredicate, což je funkce, která umožnuje omezení rozsahu exportu na vybrané části dokumentu (viz obrázek 2).



Obrázek 2: Export zákona o daních z příjmu

Předpokladem splnění tohoto úkolu bylo porozumět struktuře dokumentů a procesu jejich tvorby. Každý dokument se skládá ze stylových, jednotkových, hlavičkových a textových elementů. Textové elementy obsahují dobu platnosti a samotný text. Každému textovému elementu předchází stylový element určující specifické vlastnosti textu. Jednotkové elementy označují vždy jeden logický celek (například paragraf 56). Hlavičkové elementy jsou vždy na začátku listu a obsahují základní informace o dokumentu. Tyto elementy jsou řazeny v listu tak, aby po sekvenčním průchodu bylo umožněno sestavení daného dokumentu. Příkladový list znázorňující strukturu úvodní části Občanského zákoníku lze vidět na obrázku 3.

Obrázek 3: Struktura elementů úvodní části Občanského zákoníku

Jedním z řešení daného problému je převést nejdříve elementy do formátu DOCX a následovně převést z DOCX do PDF. Pro převedení elementů do formátu DOCX jsem využil knihovnu, která mi tvorbu ulehčí. Vybral jsem knihovnu Apache POI, která slouží jako API pro tvorbu Microsoft Word dokumentů. Pro správnou stylizaci textu jsem vytvořil šablonu v Microsoft Word, ve které vytvořím používané styly textů.

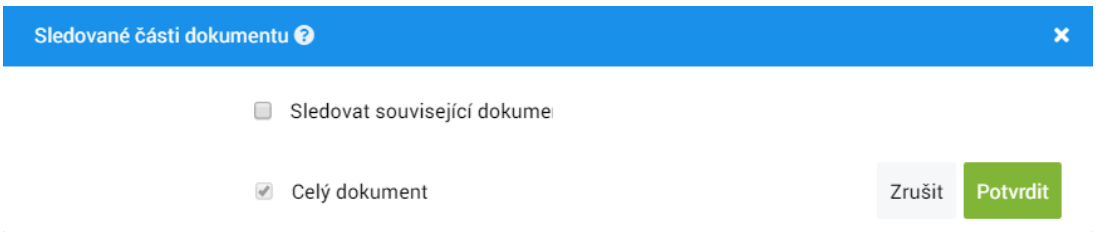
Dalším krokem bylo vytvoření unitových testů a očekávaných výstupů podle, kterých budu schopen kód pohodlně ladit. Implementaci začínám načtením šablony pomocí výše zmiňované knihovny a postupně do ní budu vpisovat texty společně s definovanými styly do XWPFDocumentu. Elementy je potřeba sekvenčně projít a pro každý typ elementu zapsat do Word Dokumentu ekvivalentní operací. Kromě samotného textu v dokumentu měl být vygenerován i klikatelný obsah (hyperlink). Nicméně, v současné verzi knihovny Apache POI nebyla implementace funkce sloužící k vytvoření klikatelných hyperlinků ještě dokončena. Navíc v dokumentaci k dané knihovně nebyly o třídě žádné informace. Pomoc jsem vyhledal na StackOverflow [10], kde mi zkušení uživatelé radili jak hyperlinky v dokumentu implementovat svou vlastní cestou.

Dokument ve formátu DOCX byl vytvořený.

Následovalo převedení z formátu DOCX do PDF. Pro tuto operaci jsem použil třídu PdfConverter. Na rozdíl od vytváření DOCX dokumentu, převedení do PDF bylo relativně lehké a rychlé, nicméně se zde po konverzi objevily nefunkční hyperlinky. Tuto chybu, která byla zřejmě na straně knihovny Apache POI, jsem již nedokázal opravit a stávající export do PDF má nefunkční obsah. Pro dokument PDF a DOCX vytvářím REST endpoint, aby frontendový tým mohl umožnit uživatelům převedený dokument stáhnout.

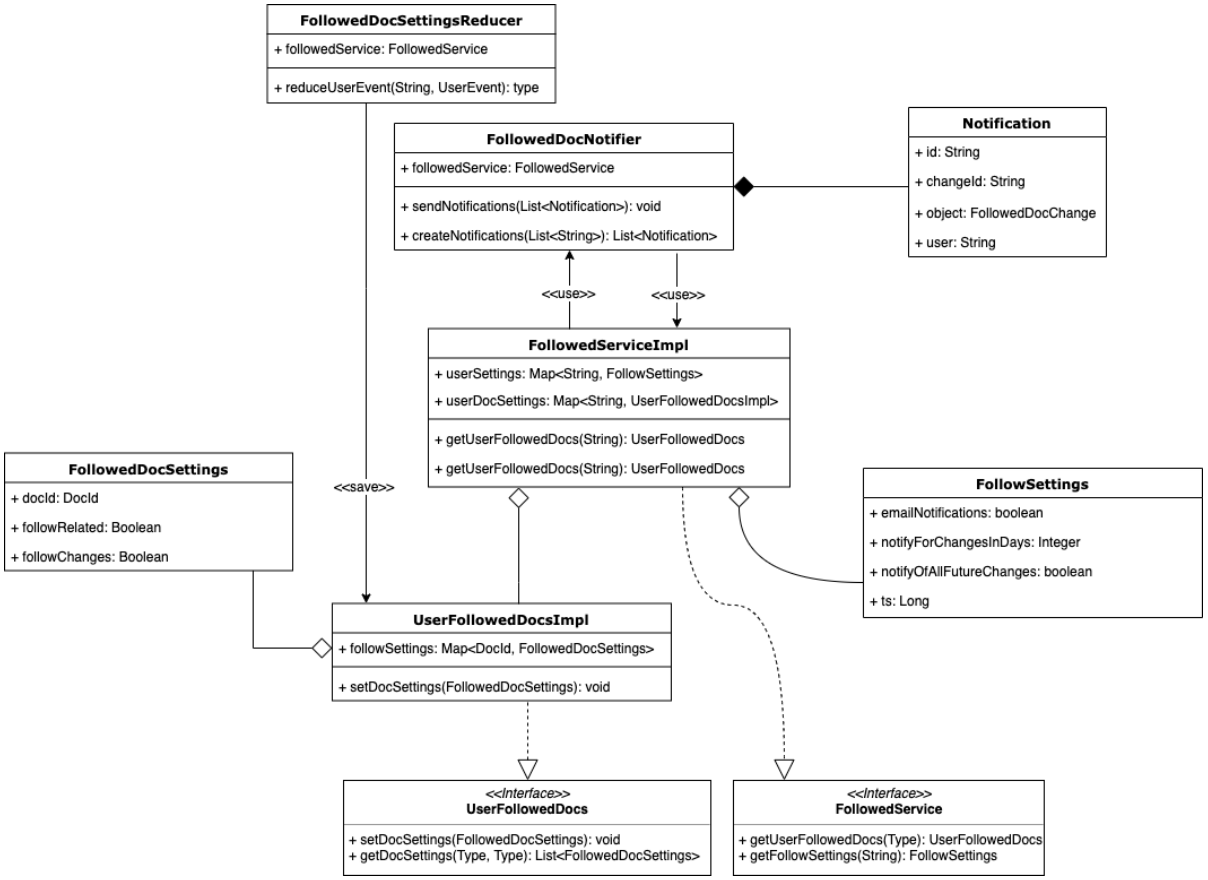
4.3 Rozšíření sledovaných dokumentů o možnost volby sledovaných částí dokumentu, volby notifikací

Jako první jsem se seznámil s implementací stávající funkcionality sledovaných dokumentů. Současná logika umožňovala uživateli sledovat změny textů a souvisejících dokumentů k dokumentu. V nastavení profilu (globální nastavení) si uživatel mohl nastavit, zda chce zasílat notifikace na email pro textové změny či pro změny souvisejících dokumentů. Uživatel si mohl nastavit i individuální nastavení pro dokumenty separátně, avšak tyto možnosti byly omezené a v podstatě nic neuměly, proto bylo třeba je rozšířit. Volbu nastavení ke konkrétnímu dokumentu před úpravou lze vidět na obrázku 4.



Obrázek 4: Volby sledování zákona o daních z příjmu před rozšířením

Stávající logika uměla jen nastavit, zda tento dokument chci vůbec sledovat a zda chci k němu sledovat i změny v souvisejících dokumentech. Vztahy mezi třídami před úpravou lze znázornit následujícím třídním diagramem (obrázek 5).

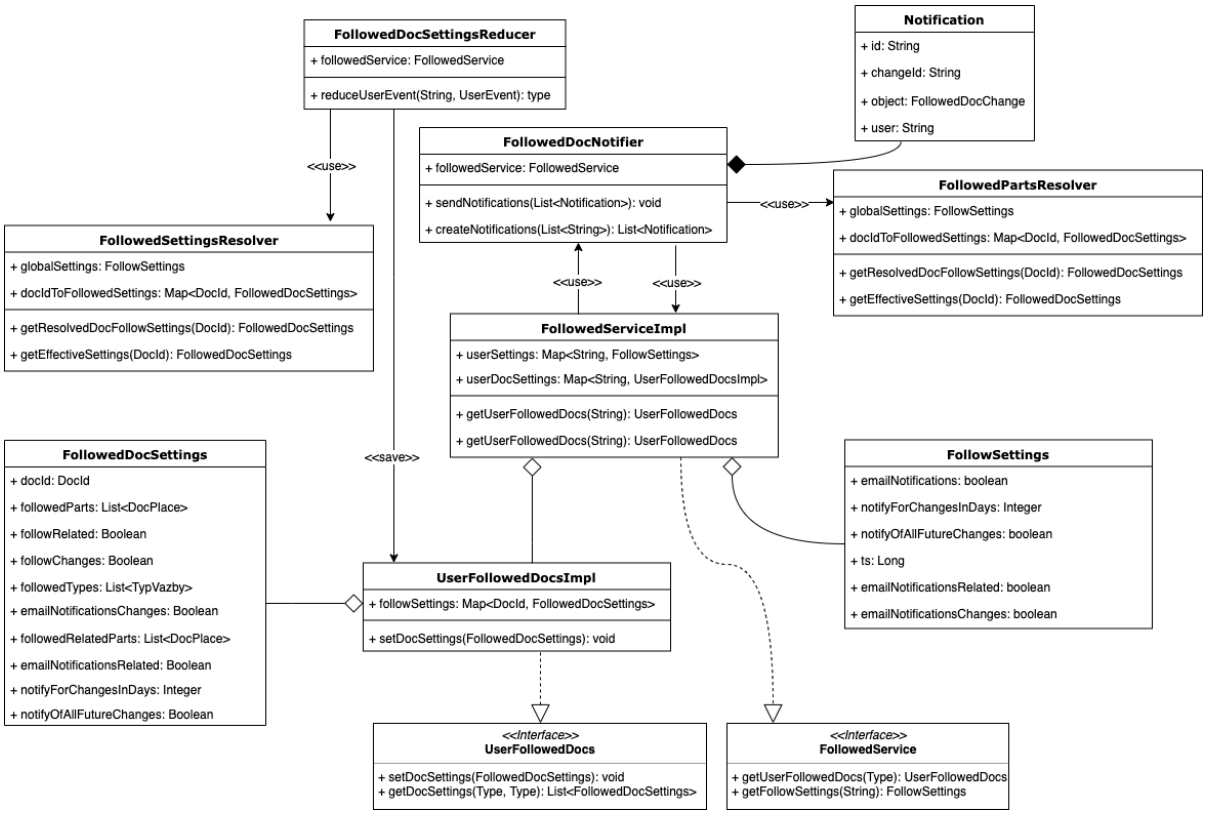


Obrázek 5: Třídní diagram před úpravami sledovaných dokumentů

Jednalo se o velice komplexní subsystém. Začal jsem implementací unitového testu pro komponentu notifikací. Jelikož konstruktor služby má parametry dalších služeb, musel jsem tyto služby v testu vytvořit pomocí Mockito Frameworku [9]. Díky této knihovně může uživatel vytvořit falešnou komponentu umějící jen to, co mu programátor nadefinuje. Začal jsem upravením algoritmu, který se staral o nastavování a ukládání individuálního nastavení na serveru. Toto nastavení se mělo propsat do nastavovacího procesu jen tehdy, pokud se lišilo od globálního nastavení. Vytvořil jsem novou třídu FollowedSettingsResolver, která umí porovnat globální a individuální nastavení dokumentu a vrátit jen to nastavení, jež se lišilo. Toto nastavení dále putovalo do nastavovacího procesu, kde se pošle na server a uloží.

Následovalo rozšíření logiky starající se o vytváření uživatelských notifikací. Vytvořil jsem novou třídu FollowedPartsResolver. Třídu jsem implementoval tak, aby uměla podle nastavení pro konkrétního uživatele vrátit jen relevatní notifikace. Pro správnou funkčnost jsem musel rozšířit třídu FollowedSettingResolver o slučovací logiku globálního a individuálního nastavení. Pokud

individuální nastavení pro dokument nebylo nastaveno, tak se vracelo globální nastavení. Toto nastavení jsem posílal jako argument do FollowedPartsResolveru filtrujícího jen relevantní notifikace pro uživatele. Výsledek všech těchto úprav a nových vztahů je znázorněn v následujícím třídním diagramu (obrázek 6).



Obrázek 6: Třídní diagram po úpravách sledovaných dokumentů

5 Teoretické a praktické znalosti získané během studia a uplatněné studentem v průběhu odborné praxe

Mezi uplatněné znalosti získané během studia patří základní znalost programovacího jazyka Java. Z předmětu Programovací jazyky I jsem uplatnil zejména parsování textu pomocí třídy Scanner, kterou jsem nejvíce využil. Další uplatněnou znalostí bylo přetížení konstruktorů. Znalost objektově orientovaného programování a dědičnosti mi pomohla lépe pochopit stávající logiku projektu. Nejvíce zužitkovanou dovedností, kterou jsem využil v přijímacím řízení do firmy, byla znalost složitostí algoritmů. V oboru informačních technologií je pro programátora nezbytnou součástí znalost anglického jazyka. Tuto získanou znalost jsem uplatnil zejména při hledání řešení problémů na webu a při práci s projektem, který je celý napsaný v anglickém jazyce.

6 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Kromě programovacího jazyku Java jsem se během mé praxe setkal s mnoha technologiemi. Mezi tyto technologie patřil například verzovací systém Git, programovací jazyk Groovy, dotazovací jazyk GraphQL, vývojové prostředí IntelliJ IDEA, síťová architektura REST, testovací framework Mockito a Cucumber.

Při nástupu na individuální odbornou praxi do firmy ATLAS Consulting mi scházeli především znalosti z programovacího jazyka Java, například streamy. Jsem si vědom toho, že v předmětu Programovací jazyky I se Java probírá včetně streamů, ale spíše byly probrány teoreticky. Další znalost, kterou jsem během praxe postrádal byla práce se známými rozhraními pro programování aplikací jako je třeba REST nebo Spring Boot. Scházela mi znalost s testovacími frameworky Mockito a Cucumber, které jsem často využíval. Dále mi chyběla znalost profesionálního a komerčního vývojového prostředí IntelliJ IDEA. Z praktických dovedností to byla zejména práce s verzovacím systémem Git [11] a s dotazovacím jazykem GraphQL, který jsme používali pro přenos dat. Navzdory tomu jsem se díky ochotným kolegům a literatuře vše doučil.

7 Výsledky dosažené v průběhu praxe a její celkové hodnocení

Absolvování bakalářské praxe ve společnosti ATLAS Consulting pro mě byla klíčovou zkušeností v mém profesním životě jako programátora. Naučil jsem se pracovat s projektem, který obsahuje desetitisíce řádků kódů. Měl jsem možnost pracovat s mnoha technologiemi, se kterými jsem se doposud nesetkal. Poznat jejich podstatu použití a jejich výhody v praxi. Získal jsem zkušenosti s programem IntelliJ IDEA, který je plně přizpůsobený k programování v jazyce Java a podporuje nejrůznější frameworky, jako je například Cucumber a Spring Boot. Dále se systémem Git, jenž funguje jako správa verzí. Systém jsem využil nejvíce pro revizi zdrojových kódů (code review). Mezi mé největší úspěchy považuji rozšíření systému o zobrazení novelizací částí dokumentů.

Výsledky dosažené v průběhu praxe byly pro mě přínosem a hodnotím je pozitivně. Rozšířil jsem své znalosti a zkušenosti v oblasti programovacích jazyků Javy a Groovy, což jsou mezi programátory nejoblíbenější programovací jazyky. Za největší úspěch považuji, že jsem do projektu přispěl novými funkcemi, které při své práci využívají reální uživatelé.

Literatura

[1] BLOCH, Joshua. *Java efektivně: 57 zásad softwarového experta.* 2. Praha: Grada, 2002. Moderní programování. ISBN 80-247-0416-1.

[2] ATLAS Consulting. [Online]. [cit. 2019-04-19] 2013. Dostupné z: https://atlasconsulting.cz.

[3] Java. [Online] [cit. 2019-19-04] 1995. Dostupné z: https://www.java.com.

[4] Apache Groovy. [Online] [cit. 2019-04-19] 2003. Dostupné z: http://groovy-lang.org.

[5] IntelliJ IDEA. [Online] [cit. 2019-04-19] 2001. Dostupné z: https://www.jetbrains.com/idea.

[6] GraphQL. [Online] [cit. 2019-04-19] 2015. Dostupné z: https://graphql.org.

[7] REST. [Online] [cit. 2019-04-19] 2000. Dostupné z: https://restfulapi.net.

[8] Cucumber. [Online] [cit. 2019-04-19] 2008. Dostupné z: https://cucumber.io.

[9] Mockito Framework. [Online] [cit. 2019-04-19] 2008. Dostupné z: https://site.mockito.org.

[10] StackOverflow. [Online] [cit. 2019-04-19] 2008. Dostupné z: https://stackoverflow.com.

[11] Git. [Online] [cit. 2019-04-19] 2005. Dostupné z: https://git-scm.com.